Reinforcement Learning: Architectures and Algorithms

Mieczyslaw M. Kokar and Spiridon A. Reveliotis

Northeastern University

360 Huntington Avenue

Boston, Massachusetts 02115

kokar@northeastern.edu

spyros@nueng.coe.northeastern.edu

1 Introduction

Constructing an intelligent *agent*, i.e., a computer system able to perform tasks that are normally attributed to humans, has been an objective for researchers in several disciplines – artificial intelligence, intelligent control, robotics – to mention just a few. The approach to building such an agent is to equip a computer system with sensors and actuators, and let it execute an infinite *sense-reason-act* loop (the main paradigm of *intelligent control*[1]). This approach distinguishes two primary entities: the *agent* and its environment, called the *world*. Typically, the world is a dynamical system whose behavior is a function of its previous state, actions exerted upon it by the agent, and time. We presume that the agent attempts to satisfy an externally defined *goal* through appropriate selection of actions. Actions are generated based upon agent's sensory inputs, goals, and its internal state (*knowledge*). The performance of an agent can be measured in terms of a *performance index*, which includes the quality of goal satisfaction and possibly the cost of the actions.

The intelligence of such an agent is incorporated in the software of its knowledge base and sense-reason-act decision procedures. These procedures associate actions with sensory inputs based on the internal knowledge of the agent. The two principal means of providing these procedures and internal knowledge to the agent are through programming and through learning. In this paper we are interested in agents with learning capabilities. In a very general sense, learning means creating knowledge structures. In this context, therefore, learning is the capability of the agent to modify its knowledge and/or decision procedures based on past experience, so that its future performance is improved.

Learning has been studied in many research communities, among them, cognitive science, control, artificial intelligence, each with its own foci. Learning agents can be classified according to the form of feedback they receive from the world in response to their previous actions.

Learning from examples (*learning with a teacher*). The learning algorithm is trained on a series of instances classified by the teacher as either *positive examples*, i.e., belonging to the concept to be learned, or *negative examples*, which do not belong to the concept. The goal of the learner is to generate a concept description that correctly classifies the seen instances and generalizes effectively as to future unseen instances. The feedback received by the learner can be interpreted as an error associated with the learner's internal representation of the learned concept; we call it *instructive feedback.*

Reinforcement learning (RL) (*learning with a critic*). The learner receives as feedback a *scalar* signal, called *reinforcement*, which provides evaluation of its performance with respect to the preset goals. The reinforcement feedback does not give any direct error information on the learner's internal representations; we call it *evaluative feedback.*

Learning by observation (*unsupervised learning, learning by discovery*). This type of learning program does not have any direct input on what it should focus its attention, which observations are positive/negative instances, or which direction to follow in search of better descriptions. The learner collects observations and derives generalized concepts according to its own internal rules.

The first method, learning from examples, requires the most external guidance. The training of intelligent agents requires an extremely well informed teacher, a requirement that is very difficult to meet. Learning by discovery, on the other hand, does not require any feedback; but the result of this is that the learner does not receive any external guidance and thus, is inefficient. This seems to be the reason that more researchers, interested in learning capabilities of intelligent agents, have recently turned to the reinforcement learning paradigm.

Our objective in this paper is to show the evolution of the reinforcement learning methods and some of the contributions of AI to this paradigm. The starting point of our analysis is the thesis that more structure is needed to make learning more efficient. We believe that this need for structure was the driving force behind the AI-based research on reinforcement learning. Therefore, this paper will focus on structural solutions to reinforcement learning developed in the AI machine learning community.

In the following section, we introduce the main ideas behind reinforcement learning. To illustrate the primary concepts of the RL paradigm we use a simple example of the maze world. We also briefly discuss the foundations of the field. In section 3, we give a formal specification of the RL paradigm. We also present a general architecture of a RL agent and its primary components. In section 4, a number of developed algorithms are discussed in the context of this architecture. The intent is to identify the motivation for developing those algorithms and to show their principal features. This includes enhancements to the basic reinforcement learning architecture, which are shown to improve the efficiency of the learner. Section 5 is devoted to architectural aspects of active perception. Finally, in section 6, we discuss possible directions and constraints for applications of reinforcement learning techniques in modern technological areas.

2 Learning Through Reinforcement

The term *reinforcement learning* has been borrowed from the area of behavioral psychology, where it has been used to describe some models of behavior-learning in humans and animals. The primary feature of those models is that they describe the behavior-learning process as a sequence of trial and error steps resulting in the formation of an action map, which defines an appropriate action for each specific situation in which the agent finds itself.

The need for reinforcement learning occurs naturally in situations where agents, similarly to humans, must derive actions (decisions) while their knowledge is incomplete and uncertain. This might be due to limited (a priori) knowledge of the agent's world, the increased complexity of the world and/or the agent which does not allow detailed analytical study of their behavior, or the extensive variation of the world and/or agent dynamics with time. As a result, the agent does not have (and it cannot synthesize) any error feedback on its internal knowledge. However, in many cases an *evaluative feedback* is available and thus can be utilized to improve the agent's future performance.

One of the consequences of the evaluative feedback is that the reinforcement learning algorithm is never certain of the correctness of any of its learned behaviors. Having a high evaluation for a behavior at a particular situation does not mean that there does not exist a better one, which has not yet been discovered. As a result, a RL algorithm must always include a search in the space of possible behaviors, in which the tradeoff between the best known behavior and the need to explore the unknown behaviors takes place. The design and analysis of algorithms to organize this experimentation efficiently and in an incremental (on-line) mode, is one of the major topic of reinforcement learning.

We will elucidate the previous discussion through the following example. Although this example is based on an agent operating in an artificial "maze world" and thus cannot be representative of the real world, we believe that it is easier to interpret than the examples coming from the psychological literature, due to the explicitness of knowledge possessed by artificial agents. Rats, the most typical subject of reinforcement learning in psychology, may have some knowledge that they use in their learning process, the amount of which and the kind are not known to us, and thus, it is difficult to analyze and evaluate their learning behavior.

Consider a mobile robot learning to navigate through the two-dimensional world[2] of Figure 1. The robot can be in any of the unmarked locations of the grid, defining the world. The position of the robot defines the world *state*. The robot is able to perceive the state of the world, and change it by performing *actions* (steps). The allowed actions are $UP, DOWN, RIGHT$, and $LEFT$ and take the robot to corresponding contiguous locations. Locations marked by an 'X' are inaccessible; they act as barriers or obstacles. If the robot selects an action which would lead into a barrier or outside of the world the state does not change. One of the locations, marked with a 'D' in Figure 1, is called the *destination*. The state of the world in which the robot has reached the destination is the *goal state*; whenever this happens, the robot is relocated to an arbitrary new state.

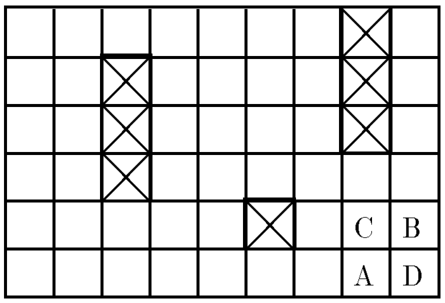

Figure 1: A world example

The learning task is to find the steps that can take the robot from any location in the world to destination using the shortest possible path. In this learning, the robot is guided by the *reinforcement*, which is either received as a special scalar input directly from the world, or is synthesized from the state input by a robot's reinforcement calculation procedure. The robot knows that it is supposed to maximize the cumulative reinforcement received over the time of its operation. There are two basic schemes for receiving the reinforcement: *immediate reinforcement* and *delayed reinforcement*. In the former case, the robot receives a reinforcement signal (say 0 or 1) depending on whether or not the step is a part of the shortest path from the robot's location to the destination. The robot then associates the returned reinforcement value with the state-action pair. In the latter case, the reinforcement signal of 1 is received by the robot whenever it enters the destination location. In all other situations the reinforcement is 0. This means that the robot is not provided with any discriminative feedback until it transitions to the destination location. This reinforcement scheme makes the task of learning much harder. In the following we show a possible strategy for learning an optimal path under the delayed reinforcement scheme.

Initially, the robot does not have any values associated with steps that it could take at particular locations. It does not even know the range of possible values of the reinforcement. It simply performs a random walk in the world. Eventually, it reaches the destination and receives a reinforcement of 1. At this moment, the robot knows that whenever it is at 'A' it can go to 'D' in one step to receive reinforcement of 1. Suppose that it has an algorithm which divides the received reinforcement by 2 (discounting) and associates it with the previous state-action pair. We refer to this procedure as *backing-up* of the reinforcement. According to this procedure, the robot associates the value of 1/2 with the pair $<$ 'A', $RIGHT >$. Suppose that during the two consecutive trials the robot first reaches 'A' from 'C' and then 'C' from 'B'. The robot associates the discounted values of 1/4 with $<$ '$C', DOWN >$ and 1/8 with $<$ '$B', LEFT >$. If the robot were always taking the best known path (*greedy policy*), it would never be able to find that there exists a better path. However, by using *exploratory search*, it will eventually find out that there is a better path B-D, since the discounted value 1/2 of the backed-up reinforcement after taking this path is greater than the best known (i.e., 1/8) before this exploration. It stores this information about the best step to be taken at location 'B' (i.e., go to 'C') and the new value of 1/2 in its internal data structure. Through this mechanism, the information about the best steps is propagated back to the states that are more distant from the goal state and eventually the robot learns shortest paths to the destination from all locations.

The formal mathematical foundations of reinforcement learning can be traced back to statistics, the theory of learning automata, and dynamic programming. The "Bandit Problem" was an attempt to cope with reinforcement learning initiated in the field of statistics; the "Tsetlin Automaton" was a similar attempt to solve the problem through the theory of learning automata[3] . Narendra and Thathachar[4] give an extended review of the work done in the field of learning automata until the mid-seventies. The close relationship between reinforcement learning techniques and dynamic programming (DP) has recently been established[5], with interesting theoretical ramifications for the field. Genetic algorithms provide a different approach to the problem of reinforcement learning. They use the mechanisms of *mutation* and *crossing-over*, inspired by biological models, to generate the set of the "fittest" production rules that define the function that maps external inputs to actions.

3 Modeling the Reinforcement Learning Problem

3.1 A Reinforcement Learning Framework

The two primary entities of the reinforcement learning framework (Figure 2) are the *learning agent* and the *world*. The world is modeled by a dynamic system whose transitions among different states are caused by the agent's actions. The agent consists of three functional parts: the *behavior* B, the *input function* I, and the *reinforcement function* R. The input function translates the world's outputs, which represent world states, into agent's inputs. The reinforcement function assigns a value to every state of the world. The behavior function updates the agent's knowledge and generates agent's actions. The above description is formalized as follows[3].

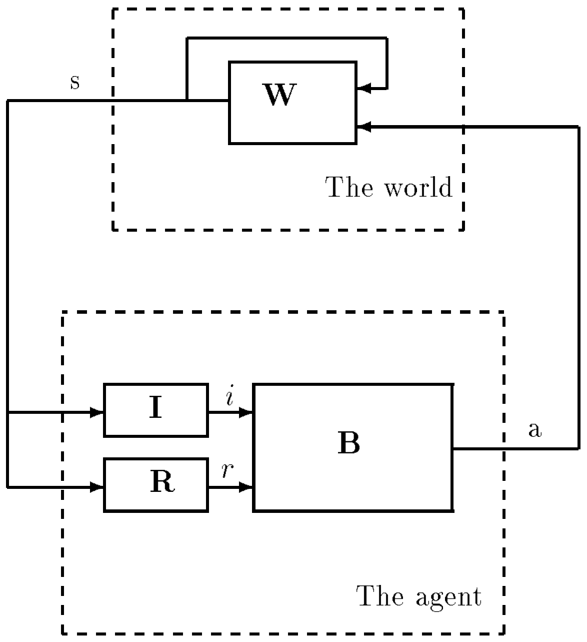

Figure 2: A reinforcement learning framework

The world is modeled as a triple

$$\mathcal{W} =< S, A, W >,$$

where S is the set of possible states, A is the set of possible actions, and W is the state transition function of the world,

$$W : S \times A \rightarrow S.$$

The agent is modeled as a 4-tuple

$$\mathcal{A} =< \mathcal{I}, I, R, B >,$$

where $\mathcal{I}$ is the set of possible inputs to the agent, I is a mapping

$$I : S \rightarrow \mathcal{I}$$

that corresponds world states to agent's inputs, R is the reinforcement function of the agent that maps states into real numbers,

$$R : S \rightarrow \Re,$$

and B is the behavior of the agent mapping strings of inputs into actions,

$$B : \mathcal{I}^* \times R \rightarrow A^*.$$

Typically, worlds are modeled by automata, either deterministic or stochastic. Designing agents that are able to deal with stochastic worlds make it possible to apply reinforcement learning methods to worlds with *apparent inconsistency*, i.e., cases where although the agent generates the same action in response to the same world's state, the resulting transitions and/or reinforcement values differ. There are a number of reasons that the world might be perceived as apparently inconsistent: stochastic behavior is an intrinsic characteristic of the world; the correspondence of world states to inputs for the agent is not one-to-one, and thus, the same input may be assigned to more than one world state; the agent's interface with the world does not function properly, resulting in misleading effects. In its most general formulation, the problem of reinforcement learning in a stochastic world is too complex to be solved. Most of the algorithms discussed in this paper work under the assumption that the world is *globally consistent*, which means that the *expected* value of the reinforcement given input i and action a remains constant over time.

The problem of learning how to reach the goal state(s) with a minimum number of actions, can be restated within this framework as the problem of adjustment of the agent's behavior so that the reinforcement it receives over a prespecified time period is maximized. In the case of autonomous agents acting over a long time in an unsupervised mode, this time period can be considered infinite. To satisfy the learning goal, the agent keeps updating its behavior towards the world, based on the received reinforcement. Ideally, the result of the updating is an optimized *action map*, i.e., a function (in the general case this is a relation) which assigns actions to incoming input strings in such a way that cumulative reinforcement is maximized.

3.2 The Learning Behavior

The general mechanism according to which the agent adapts its behavior to the incoming information from the world, i.e., to reinforcement values and (possibly) world-state relevant information, is called a *learning behavior*. Its general scheme is presented in Figure 3. The behavior consists of three parts: the *internal state* x, the *update function* u, and the *evaluation function* e. The internal state x expresses (summarizes) the level of knowledge that the agent possesses about the world; it does not relate explicitly to the world states. At every cycle, the evaluation function e determines the agent's response a to the received input i, based on the internal state x,

$$e : X \times \mathcal{I} \rightarrow A.$$

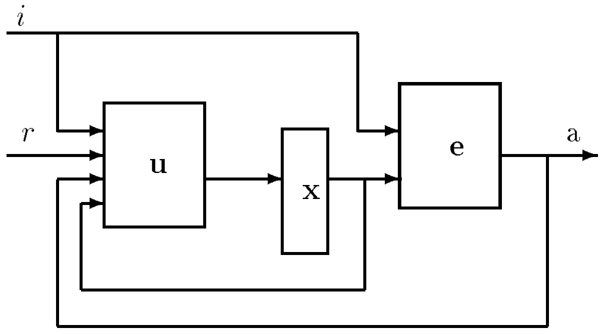

Figure 3: The learning behavior scheme

The selected action brings the world to a new state, which results in a new input i'

and a reinforcement value r. The agent's internal state x is then updated by the update function

$$u : \mathcal{I} \times X \times A \times R \rightarrow X$$

so that it better approximates the agent's knowledge about the world.

4 Implementations of the Learning Behavior: Variations and Extensions

Having described a general mechanism of reinforcement learning behaviors, we can consider specific implementations of this mechanism in a number of algorithms known in the literature. The algorithms have been selected as representatives of some broader classes of algorithms in the field. Each of them is the result of a different implementation of, and/or extension to, the general scheme. In the presentation of the algorithms, the emphasis will be on the way in which they implement the various parts (x, u, e) of the general mechanism, on the way they address the learning efficiency problems through architectural variations, and on their extented applicability.

4.1 Probability-Vector Algorithms

These algorithms have been developed by the learning automata community[4]. The main characteristic of these algorithms with respect to the general framework discussed in the previous section, is that in their evaluation and update functions, they do not use information about the state of the world; the only input to these algorithms from the world is the reinforcement.

At every step, the agent selects an action according to a probability distribution. This distribution is stored as a probability vector p, which associates with every action a_j the probability value p_j that this action will be selected. In terms of the learning behavior described in the previous section, the internal state x of the agent is the probability vector p, and the evaluation function e is based solely on these probabilities. The update function adjusts these probabilities based on the actions performed on the world and on the returned reinforcement. The resulting agent is a *stochastic learning automaton*.

As a characteristic example, consider the *Linear Reward-Penalty (L_{RP})* algorithm[4]. To update the probabilities in response to an action a_j and positive reinforcement, this algorithm first decreases the probability values of all the other actions by an amount which is proportional to their current values, and then it updates the probability p_j associated with action a_j so that the sum of all probability values adds up to unity. When the reinforcement is negative, the respective probabilities are increased instead of decreased.

In a variation of the L_{RP} algorithm, known as *Linear Reward-Inaction (L_{RI})* algorithm, probability values are updated only when the agent receives positive reinforcement after taking an action. Although this algorithm exhibits a significant structural similarity to L_{RP}, the two algorithms have significantly different behaviors in terms of convergence and overall performance. Many other versions of these algorithms have been obtained by using different nonlinear updating policies. They have been proposed to improve learning performance in particular applications.

Another algorithm that can be classified in this category, since it uses only reinforcement information received from the world, is the *Interval Estimation (IE)* algorithm, developed by Kaelbling[3]. This algorithm's internal state consists of the upper bounds of the confidence intervals of positive reinforcement received in response to actions a_j. The algorithm utilizes statistical methods to update these bounds. More specifically, for every action in the agent's action set, the returned reinforcement is monitored, and from the collected data, the upper bound of a confidence interval of the probability of receiving positive reinforcement in response to this action, is estimated. In the evaluation function, the action with the highest upper bound is selected. The upper bound of the confidence interval for a particular action may be high either because this action has a high probability of receiving positive reinforcement or because the collected data is insufficient for an accurate evaluation. In this way, the trade-off between following the greedy policy and performing further exploration is done automatically. The level of confidence for which the intervals are estimated is predefined; it constitutes a tuning parameter of the algorithm. This algorithm is conceptually simple, yet it has been shown to outperform many other algorithms from the same class[3].

The main problem with the probability-vector algorithms is that they are designed to search for the globally best action based upon received reinforcement, and are thus they not able to associate a locally optimal action with each state of the world. Consequently, these algorithms either continuously change their internal state, i.e., are unstable in terms of convergence to optimal behavior, or when they are designed to enforce convergence, they may get stuck in nonoptimal states (*absorbing states*).

4.2 Associative Algorithms

In contrast to the previous class, associative algorithms take into consideration not only the reinforcement returned by the world, but also information about its internal state perceived through the input function I. They *associate* action-probability vectors with the states. A trivial way to solve this problem would be to use a separate stochastic learning automaton for each state. However, such a representation would not allow the association of one action-probability distribution with a set of states. In other words, such a representation would not allow for generalization. This problem was solved[6] by using parameterized classes of distribution functions to represent action-probabilities. Under this representation, learning the optimal behavior is equivalent to learning an optimal set of parameters for the distribution functions.

Associative RL algorithms are most typically implemented in a *neural net* architecture. A neural net consists of layers of processing units, including an *input layer*, an *output layer* and possibly a number of *hidden layers*. The input vector i is passed to the first layer for processing; the output from each layer is passed to the consecutive layer for further processing. Each unit within a layer, multiplies its input vector by the weighting vector associated with this unit and filters this result through a decision making function to produce an output for the next layer. The outputs from the network are interpreted as parameters of single-parameter action-probability distribution functions. The evaluation function of an associative RL agent uses these distributions to generate actions. The probabilistic interpretation of the evaluation function provides the exploration in the search for better behaviors of the associative RL algorithms.

The internal state s of the agent consists of the network's weights. The update function updates the weights based on the received input vector i and reinforcement. The updating process can be interpreted as *stochastic hill-climbing*, i.e., an incremental movement in the weight space from the current point towards the steepest increase of the expected value of the returned reinforcement$x$.

Since the focus of this paper is on the structural aspects of the reinforcement learning algorithms rather than their hardware implementations, we present a rather simple, but quite representative associative reinforcement learning algorithm developed by Sutton[8]. It is called *Linear-Associator Reinforcement-Comparison* algorithm *(LARC)*. A distinguishing feature of *LARC* is that in addition to the returned value r of the reinforcement, it also uses a predicted value p, and that the updating mechanism uses the difference $r - p$ to determine the direction and degree of change for the values of the weights. This kind of algorithms are called *Reinforcement-Comparison* algorithms.

The implementation of *LARC* consists of two networks. Both of them receive the same input vector i, consisting of external signals describing the world state. The first network computes the predicted value p of the reinforcement as the inner product of the input vector i and its weight vector v. The second network implements the evaluation function. It computes the inner product of its weight vector w and the input vector i, adds a random number n to the previous result, and selects one of two possible actions (producing an output of 0 or 1), depending on whether the resulting sum is greater or less than a threshold θ. The update function u adjusts the components v_j and w_j of the two weight vectors on the basis of the action taken at the previous step and the difference between the received reinforcement and the predicted reinforcement $r - p$.

$$\Delta w = \alpha(r - p)(y - 1/2)i$$

$$\Delta v = \beta(r - p)i$$

The coefficients α and β in the above formulas are called *learning rates*. Setting a high value for the learning rates may increase the speed of convergence of the learning algorithm, but it may also increase the probability that the algorithm becomes unstable or gets stuck in a locally optimal solution.

By utilizing classification abilities of neural nets, associative algorithms are able to associate optimal actions with whole classes of the world states, or in other words, they are able to *generalize* from their past experience with interactions with the world. This ability is a very important feature for learning algorithms with respect to time and space complexity. However, like many connectionist algorithms, they converge rather slowly and are unstable when applied to more complicated learning tasks. Furthermore, the distributed representation used by these algorithms, although an effective generalization mechanism, makes it very difficult to interpret the learned structure and to prove its validity; the learned concepts are represented by weight vectors, which do not have any direct meaning.

4.3 Learning with Delayed Reinforcement

All of the algorithms described above work under the assumption that the world returns a reinforcement value in response to every single action taken by the agent. However, there are a number of application tasks in which the agent receives evaluation of its behavior following an entire sequence of steps (for example, the problem shown in Figure 1). This kind of reinforcement scheme is called *delayed reinforcement*. In this section we discuss how the problem of learning with delayed reinforcement can be solved using variations of dynamic programming.

Dynamic programming (DP) techniques compute the optimal action map, given a complete state transition model of the world and the credit (in our case, reinforcement) associated with each transition. In their computation they use the notion of *state values*. The value $e(s)$ of state s is associated with an action map μ and defined as the cumulative reinforcement collected by the agent provided it starts from state s and follows the action map μ thereafter. Since the actual value of cumulative reinforcement is not known, it is estimated as the sum of the immediate reinforcement received after executing a single action, plus the current value of the state resulting from this action. The optimal action map maximizes the value of every state. DP techniques compute the optimal value for each state s by iteratively comparing the estimated cumulative reinforcement for all possible actions at state s and replacing the state value $e(s)$ with the maximum. We say that the state values are *backed up* during these iterations. The optimal action map assigns to every state the action that was used in the backing-up of the optimal state value.

A variation of the previous backing-up algorithm, proposed by Watkins$x$ keeps a value Q_{sa} for every state-action pair, in addition to the state value $e(s)$. For a state-action pair $< s, a >$ and an action map μ, Q_{sa} is defined as the cumulative reinforcement collected by performing action a while in state s, and following the action map μ thereafter. In this approach, the value of a state s, $e(s)$, is defined as the maximal Q_{sa} value over the set of possible actions for this state. Watkins has proven that if, in lack of knowledge of a complete world model, the Q_{sa} values are updated according to the formula

$$Q_{sa}^{k+1} = (1 - \beta_{sa}^{k})Q_{sa}^{k} + \beta_{sa}^{k}[r + \gamma^{k} e(succ(s,a))]$$

they will converge to the optimal ones, provided that each transition is tried an infinite number of times, and $\beta^k \rightarrow 0$, when $k \rightarrow \infty$. In this formula, $e(succ(s,a))$ is the value of the state resulting from action a, β^k is the learning rate of the algorithm, γ^k is the discounting factor of the future reinforcement, and r is the immediate reinforcement. Since this result allows for updating only the Q_{sa} value of the current action, the updates can be done in real time. The resulting algorithm is known as *Q-Learning*.

In this section we describe an extension of the Q-Learning algorithm, developed by Sutton[3], which is known as *Dyna-Q+*. *Dyna-Q+* introduces two major extensions to the Q-Learning algorithm. The first one is an enhancement of the exploration component of the algorithm. Specifically, the value assigned to a state-action pair is not based only upon the expected cumulative value of the reinforcement received after taking this action (transition), but also includes a term which expresses the uncertainty related to this Q_{sa}-value. Sutton calls this term *exploration bonus*, and defines it to be a very small fraction of the square root of the elapsed time steps n_{sa} since action a was tried in world-state i. The introduction of this term in the credit estimation renders the algorithm sensitive to possible structural changes taking place in the world.

The second extension is of greater architectural interest, as it demonstrates an integration of world modeling with reactive reinforcement learning techniques. The learned model of the world is used for more systematic planning of future actions. This results in an increased speed of convergence of the Q-Learning algorithm. The *Dyna-Q+* architecture is presented in Figure 5. The major difference between this architecture and the general scheme presented in Figure 1 is the introduction of a *world model*. In this

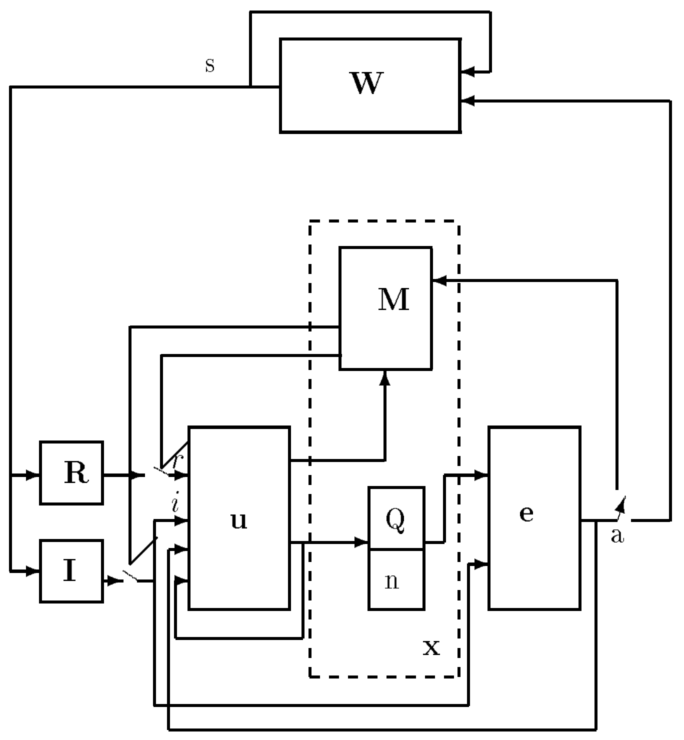

Figure 4: The *Dyna-Q+* architecture

implementation the model is a look-up table indexed by the state-action pairs. It provides the next state and reinforcement values, given an action and the current state. This table is filled incrementally every time a new transition (state-action pair) is executed. The information in the model is used for learning purposes between every two successive interactions with the world. State-action pairs and associated reinforcement values, which have been previously experienced and stored in the model, are randomly selected and passed as input to the learning algorithm. The net effect of this technique is a faster convergence of the Q_{sa}-values, because the reinforcement information recently experienced by the agent through its interaction with the world is *safely* propagated to the rest of state-action pairs during those "dummy" iterations. Sutton also suggests[2] that the modeling technique can be extented to cover stochastic behavior of the world, and that in the case of nontrivial world structures, special strategies for selecting the state-action pairs could be used in the learning steps (*search control*).

The state x of the learner is implemented in this algorithm by the Q_{sa}-value matrix, the values of the n_{sa} matrix, and the knowledge captured in the world model. The updating function is implemented by the Q-Learning algorithm with the maximal-credit estimator, modified to incorporate the exploration-bonus term. The evaluation function probabilistically selects the next action according to the Boltzmann distribution.

$$pr(a_i|s) = \frac{e^{Q_{sa_i}/T}}{\sum_{a_j \in A} e^{Q_{sa_j}/T}}$$

According to this formula, the action with the highest Q_{sa} value has the highest probability of being selected. The parameter T of the distribution regulates the degree of exploration in the algorithm. A smaller value of T means a closer following of the greedy policy.

4.4 Model-based Algorithms

The algorithms discussed in previous sections are *reactive* or *direct* in the sense that they learn to relate incoming information concerning the world states directly to actions executed by the agent. Another approach to learning how to act in a world consists of building *internal representations* of the world, i.e., *models*, and then utilizing the learned models to select appropriate actions. In this approach, the goal of learning is to generate such models that give the best approximation (best fit) to the description of the behavior of the world. Since the world is viewed as an automaton, the models should be able to correctly predict the transitions of the world for given initial states and applied actions.

Similarly as in associative algorithms, input to the learning agent includes world states. However, it does not include explicit reinforcement. The value of reinforcement is calculated by the reinforcement function R (Figure 2). Since the selection of an appropriate action is strongly dependent on the correctness of the model, the reinforcement value is a measure of how well the working model *fits* (predicts) the real world behavior. The internal state x of the learning behavior encodes a model of the world.

Models created by the agent can be of two kinds: *parametric* or *non-parametric*. Parametric models describe the world behavior in terms of parametrized procedures (equations) which can be used to calculate the world's transitions given an initial state and an action. The learning task is the adjustment of the parameter values so that the transitions predicted by the model are as consistent as possible with the actual transitions observed in the world. Non-parametric models, on the other hand, do not utilize analytical equations. Instead, they consist of a number of initial-state/action/final-state triples – instances of the world transitions. For worlds with infinite number of states, the continuity property of the world is utilized for predicting world transitions. In the prediction procedure the methods of interpolation and extrapolation are used. Learning of non-parametric models consists of either adding more transition instances to the model, if its predictive power is not satisfactory, or removing redundant transition instances, if they can be interpolated from the remaining points of the model.

In the following we present an example of a reinforcement learning system which learns multiple non-parametric models in the context of a restructurable control application[9] (Figure 4). In this kind of application, the goal is to control the value of an output variable of a plant which undergoes structural changes in its behavior. Structural changes in the plant's behavior mean that it is described by different models at different times (different sets of variables and relationships). Since the plant's behavior can change in an unpredictable fashion, no assumption can be made about the model equations, and thus non-parametric representation of models is well suited for this kind of application. The agent must learn the possible plant behaviors from observational data. Due to the high computational cost of learning, when the plant switches to another behavioral pattern the learned model is not discarded, but is stored in the agent's database, so that it can be used if the plant returns to this behavior at some time in the future. Thus, in addition to learning correct models for the plant behaviors, the agent must also learn to select the appropriate model from the model database. The system presented in this section[9] learns such a selection policy.

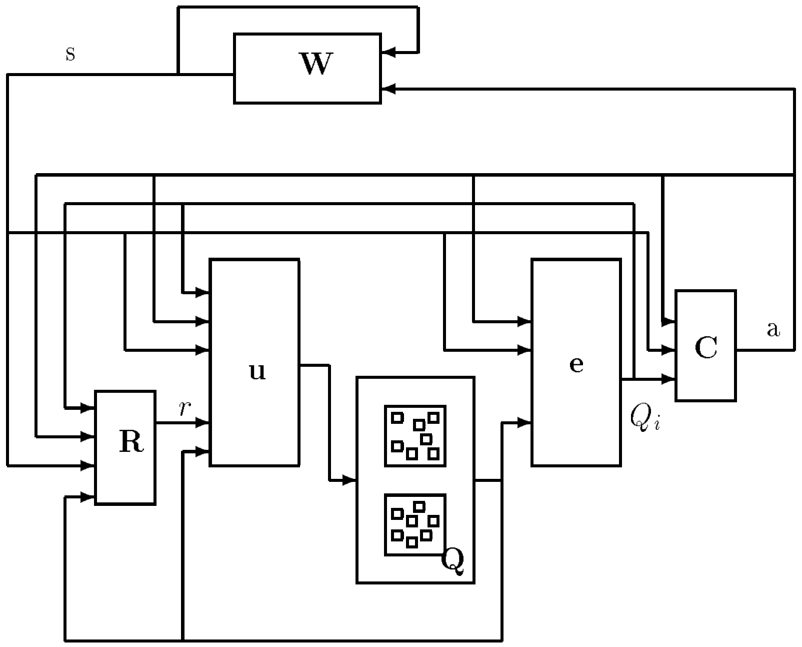

Figure 5: A multiple non-parametric model-based RL architecture

The central data structure implementing the internal state x of the learner is the database of state transitions of the plant, Q. This database consists of a number of blocks $Q = \{Q_1, \ldots, Q_n\}$, with each block being associated with a different qualitative behavior of the plant (i.e., with one model). More specifically, an element q of a block Q_i is an instance of the plant's state transition function represented by the previous state $s_0 \in S$, the next state $s_\tau \in S$, the value of action $a \in A$ exerted upon the plant during the time-interval $[0, \tau]$, the value of the time interval τ, and the parameters of the i-th model P_i

$$q =< s_0, s_\tau, \tau, a, P_i > .$$

In the discussed system, generalization over the state transition instances is implemented through two mechanisms: interpolation and the principle of physical similarity. First of all, only some of the transitions are stored; transitions which can be recovered through interpolation are discarded. The principle of physical similarity allows for grouping of similar transitions into classes; the whole class can be generated from one representative using this theory.

The evaluation function selects the *working model* from the model database on the basis of the accuracy with which the existing models predict the observed plant behavior. The update function has a dual role: it must adjust the evaluation function so that it selects the correct model for every state of the plant, and it must also adjust the selected (working) model so that its predictive accuracy is increased. In case that none of the existing models is judged appropriate for the description of the current plant behavior, the update function may initiate the creation of a new plant model.

The main advantage of this kind of algorithm is that the learned models are stored in the database. Thus the effort of re-learning the models when the world returns to a previously learned behavioral pattern is avoided. In such a case, the previously learned model is retrieved. Parametric models represent knowledge in a concise form and therefore require less memory. One of the disadvantages of such models is the need to search for the most appropriate models in a potentially large database. In addition, such algorithms should have a function that determines whether a given model should be updated or another model should be selected.

5 Reinforcement Learning Through Active Perception

There is an implicit assumption in the algorithms discussed in the previous sections, that the world inputs given to the learning agent include complete information about the state of the world. Although this looks like a natural requirement from a theoretical point of view, it is unnecessary or impossible in many practical situations. Humans, for instance, act in the world with only partial information about its state. Not only would such complete information be unnecessary for making decisions about actions, but it would also overload the information processing system, both in terms of memory capacity and computational time requirements. A possible solution is in selective acquisition of only *relevant* information about the world.

To implement a reinforcement learning system able to deal with the problems of time and space complexity resulting from the richness of the world state space, Whitehead and Ballard[10] have proposed an extension of the basic architecture presented in Section 3. The main idea behind this architecture (see Figure 6) is to incorporate an active sensory-motor system able to focus the attention of the learning system on inputs relevant to the decision making process; the authors call this perceptual scheme *active perception.* Instead of defining the input space, $\mathcal{I}$, of the agent in such a way that it captures every detailed piece of information about the world, indexical representation registers only such information about the key (*indexed*) objects that is relevant to the current task. Such an approach leads to a more compact input space for the agent, but demands the integration of an active sensory-motor system in the overall reinforcement learning framework.

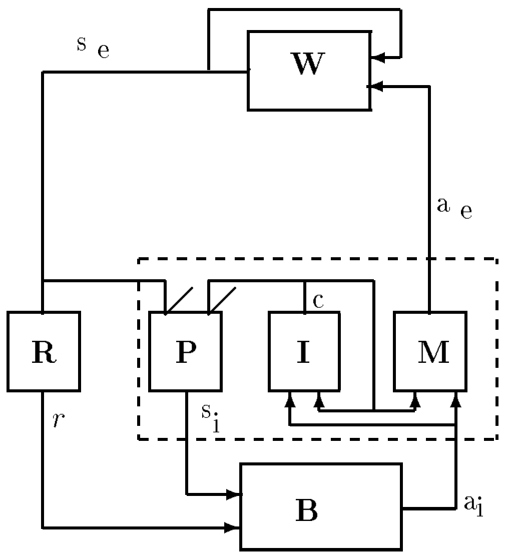

Figure 6: An active perception RL architecture

In this particular architecture, the world is modeled in the same way as in the framework presented in section 3. However, the agent model is broken down explicitly into three components: (1) the sensory-motor subsystem, (2) the reward (reinforcement) center, and (3) the decision subsystem. The novel part is the sensory-motor subsystem, whose distinguishing feature is that it is *active*, i.e., it is able to change its *internal configuration,* C, of the sensors and actuators. The sensory-motor system consists of a perceptual function, P, mapping world states S_e into agent's internal representations S_i, according to its internal configuration C,

$$P : S_e \times C \rightarrow S_i,$$

and a motor function M, mapping (transducing) agent's decisions (internal motor commands) A_i into external actions A_e,

$$M : A_i \times C \rightarrow A_e.$$

The sensory-motor system is a dynamic system with the state transition function

$$I : A_i \times C \rightarrow C.$$

The reward center R composes the returned value of the reinforcement r from the perception of the current state of the world $s_e \in S_e$ in a fashion similar to the one described in the discussion about learning world models. Finally, the decision subsystem B corresponds to the learning behavior of the framework of Section 3, with the incoming/outgoing signals being received from/sent to the sensory-motor subsystem.

The space of internal representations is considerably reduced by focusing attention only on a limited amount of the state information. This gain, however, does not come for free. Under the indexical representation, the correspondence of world states to internal inputs becomes many-to-many, destroying the global consistency of the world. This is called *perceptual aliasing*[10]. Since perceptual aliasing introduces local maxima in the evaluation function, it thereby renders the reinforcement learning algorithms previously described inappropriate for application in this architecture. These tend to confuse the learning behavior in its search for the optimal policy. This problem is solved by introducing an internal cycle in the learning behavior[10]. This cycle, called *perceptual cycle*, identifies the correct (unambiguous) representation for the perceived world state from an entire set of candidate representations.

The distinguishing feature of the active perception approach is that, in addition to learning how to act correctly in the world, the agent also learns to focus its attention on the objects which are relevant to a specific task. Preliminary results of experiments with this architecture[10] are promising. Tasks at which the general algorithms presented in earlier sections failed have been successfully completed using this approach.

6 Applicability of Reinforcement Learning

Although considerable progress in theoretical work on reinforcement learning algorithms and architectures has been achieved, there is much to be done in the area of application of these methods to real-world problems. The problems with the applications can be attributed mainly to the high complexity of the learning algorithms, both in terms of time and memory, and to the fact that in the general case, they converge very slowly to the optimal behavior, especially if the set of possible states of the world is large.

Time complexity is a very important factor in the evaluation of reinforcement learning algorithms since reinforcement learning is an iterative *real-time* process. A reinforcement learning algorithm performs the same computation in every loop and, therefore, its time complexity issue can be reduced to the time complexity of a single iteration. Thus, in designing an RL algorithm, it is essential that every iteration remains bounded . Accepted values for these bounds vary from application to application. They should be estimated on the basis of the time constraints of a specific application. Most of the update and evaluation functions of the learning behavior involve processing the entire volume of information stored in the agent's database. As a result, setting bounds on the space for storing agent's data is as important as bounding the execution time of each iteration. In the reinforcement learning literature, an algorithm that keeps bounded its time, as well as its space needs, is called *strictly incremental*. It should be made clear, however, that many of the existing algorithms do not obey such a restriction.

A great deal of theoretical analysis has been performed on the asymptotic behavior of the RL algorithms, mostly for stationary worlds. Algorithms have been classified with respect to how nearly they ultimately achieve the *optimal* behavior, i.e., the behavior that maximizes the returned reinforcement over the considered time horizon[3,11]. However, for real-world problems, optimal behavior may not be easily identifiable. This is especially crucial in the case of time-varying worlds where the world changes its behavior, sometimes drastically, so that the assumption of stationarity is not justified. As a consequence, the quality of the *transient* behavior of these algorithms is critical.

In spite of all of the abovementioned problems, the field of reinforcement learning has recently experienced a resurgence of interest by researchers. There are many different for this turn. For one, the computational power available through modern computer technology, in terms of both execution rate and storage capacity, has considerably increased. The advent of parallel computation seems very promising for RL algorithms, since many of them have an inherently parallel structure.

A second reason comes from the kind of problems in modern applications. For instance, many of them involve the control of large complex physical, technological, or even socio-economic systems, whose structures are dynamically-varying and intractable in terms of more classical control paradigms. Learning control is the most powerful modern tool for dealing with high levels of uncertainty. Even if the application of RL does not result in an optimal behavior of the controlled plant, due to the large-scale nature of these systems and/or the very low error-tolerance allowed for the controlled processes and their results, even small improvements of the system's overall performance can be economically significant.

As an example of the application of reinforcement learning in an otherwise intractable problem, consider the high-precision control of a robot manipulator[1]2 . Typically such a system has strong local nonlinearities which cannot be modeled and compensated by standard feedback control techniques. These nonlinearities severely affect the accuracy of the positioning and/or motion of the end-effector. Since in many robotic applications this accuracy is of primary importance, the application of RL control, on top of the conventional (feedback) controller, is highly justified. In this combined scheme, the control generated by a conventional feedback controller is adjusted by the RL controller in order to compensate for local nonlinearities.

Narendra[11] discusses application of RL in large-scale systems. These systems quite often consist of a large number of semi-autonomous components performing a limited number of functions. Typical examples are communication networks, computer networks, and economic networks. Theoretical analysis and simulation have shown that RL techniques can be applied for the control of these systems in a distributed fashion. A RL controller can be developed for each node of the network. Each controller is ignorant of the existence of the other controllers and their behaviors. However, the reinforcement value returned by the world is based on the overall (net) effect they have upon the system. The analysis of such a scheme falls in the area of game theory. An interesting result of the study of these systems is that whatever learning behavior is applied, the overall performance of the system, in terms of returned reinforcement, is improved compared to random behavior. Economies of scale make any prospect of performance improvement look significant in this kind of large system.

Finally, a very promising step for future development of the field, in both theoretical results and realistic applications, is the fact that RL researchers have made significant attempts to summarize and organize the work carried out in the field during the past decades. A more formal, theoretical approach is being developed. This approach allows a more systematic study of a number of interesting aspects of the RL task, like concurrent learning of policies satisfying multiple goals, cooperation and transfer of knowledge among a group of learning agents, as well as conditions and schemes for effective decomposition of the learning task into a number of simpler tasks. The relationship of RL to other scientific disciplines is being analyzed, and comparative studies between RL algorithms and algorithms developed in other areas are being carried on. The merits and pitfalls of RL algorithms are becoming clearer, highlighting the applicability and limitations of these methods to contemporary control applications. They also set the problems and goals for future research in the field. In this paper, we have given a general framework for theoretical analysis of the developed algorithms, and outlined the major issues of future research as they emerge from current theoretical studies and applications in RL.

Acknowledgements

This work has been partially funded by the NSF grant IRI-8915057.

References

[1] H. E. Stephanou, A. Meystel, and J. Y. S. Luh. Intelligent control: From perception to action. In *Proceedings of the IEEE International Symposium on Intelligent Control*, 1988.

[2] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynaming programming. In *Proceedings of the 7th International Conference on Machine Learning*, pages 216–224, 1990.

[3] L. P. Kaelbling. Learning in embedded systems. Technical Report TR-90-04, Teleos Research, CA, 1990.

[4] K. S. Narendra and M. A. L. Thathachar. Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 4:323–334, 1974.

[5] C. J. C. H. Watkins. *Learning with Delayed Rewards.* PhD thesis, Cambridge University, Psychology Department, 1989.

[6] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

[7] R. J. Williams. A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *Proceedings of the IEEE First Annual International Conference on Neural Networks*, pages 601–608, 1987.

[8] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, Amherst, MA, 1984.

[9] M. M. Kokar and S. A. Reveliotis. Integrating qualitative and quantitative methods for model validation and monitoring. In *Proceedings of the 1991 IEEE International Symposium on Intelligent Control*, pages 286–291, 1991.

[10] S. D. Whitehead and D. H. Ballard. Active perception and reinforcement learning. In *Proceedings of the 7th International Conference on Machine Learning*, pages 179–188, 1990.

[11] K. S. Narendra. Large stochastic systems: Learning automata. In Singh M., editor, *Systems and Control Encyclopedia: Theory, Technology and Applications*, pages 2714–2719. Pergammon Press, 1987.

[12] J. A. Franklin. Refinement of robot motor skills through reinforcement learning. In *Proceedings of the 27th Conference on Decision and Control*, pages 1096–1101, 1988.